Team-Oriented Process Programming[1]

Technical Report UCI 91-68

August 28, 1991

Patrick S. Young and Richard N. Taylor
pyoung@ics.uci.edu taylor@ics.uci.edu

Department of Information and Computer Science
University of California, Irvine, CA 92717

Abstract

Team-oriented process programming promises to provide significant support for the planning, directing, and controlling of software engineering projects. In this paper we apply process programming to software engineering teams and show how this can provide powerful new capabilities for the management of software projects. We identify key issues which must be addressed to apply process programming to teams, and present our vision for team-oriented process programming.

1. This material is based upon work sponsored by the Defense Advanced Research Projects Agency under Grant Number MDA972-91-J-1010. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1. Introduction

Software projects involve teams of humans, all working together for a common purpose. Managing large software projects is complex, often exceeding our human capabilities. By harnessing the computer, however, we can enhance our innate capabilities. In this paper we apply process programming to teams and show how this can provide powerful new capabilities for the management of software projects. Team-oriented process programming, especially when combined with other domain-specific process programs, promises to provide significant support for the planning, directing, and controlling of software engineering projects. It should also provide support for organizing and staffing the software engineering project. Individual team members also benefit as team-oriented process programming can increase their ability to manage and control their own parts of a project.

In this paper we discuss how process programming might be applied to teams. We identify the important issues in team-oriented process programming and discuss some potential solutions mechanisms. We briefly note how team-oriented process programming might aid project management and team members then conclude by showing a vision of how a team-oriented process programming system might look.

2. Relationship to Current Work in Process Programming

Process modeling is an active and growing field of study within the software engineering community [SPW4 1988], [SPW5 1989]. A great deal of interest has been expressed in the development of formalisms or process programming languages which allow actual enactment of the process models.

Our work focuses on the development of process programs for software engineering teams. We focus on the human processing agent, treating the team members as the processors executing the process program rather than as outside agents communicating with the program via input/output operations. We have adapted a viewpoint of the process similar to that taken by tactical management, and we envision the team manager, rather than an outside expert, as the process programmer. We are interested in the assignment of activities to team members and in the communications between team personnel. We are much less interested in how team members complete their individual assignments. We expect to handle individual assignment processes by interfacing our team-domain oriented process programs with process programs developed for other specific domains.

In addition to the work of the process modeling and process programming communities, our work has been influenced by research on teams (for example [Curtis et.al. 1988]) and on computer-supported cooperative work (see, for example, [Grudin 1988] [Olson 1989]).

3. Process Programming Applied to Team-Oriented Processes

In this section we analyze the human processing agent to identify some of the requirements on team-oriented process programming. We also compare team process programs to application programs and discuss how their differences affect the design of a team-oriented process programming system and language.

3.1. The Human Processing Agent

Team-oriented process programming can be understood through an analogy: the workers executing the software development process correspond to the hardware processors executing an application process. In some ways this correspondence works quite well: both humans and computers are computing devices. However, each has its own strengths and weaknesses. In this subsection we study many of the ways in which the software engineering team differs from a multi-processor computer.

3.1.1. Processor Differences

Human beings are individuals each with their own special qualities and abilities. In contrast, the processors in a typical multi-processor machine all have the same capabilities. While most application programs are written without regard for which processor will execute each activity, this will not be true of a good program written for execution by human beings. Consider for example a process program to write, test, and release a flight reservation system. Team members will probably vary widely in their ability to break the system into components, write code, write coherent documentation, and analyze algorithms to develop test cases. Team members will also differ in their experience with similar systems. One team member may have worked on travel agency software before, another may have experience with databases, while a third may have worked with the particular terminals and printers used in the project. A team-oriented process programming language needs a method for specifying either a particular processor or the necessary capabilities of a processor for each activity.

3.1.2. Learning and Skills Acquisition

Humans also have the remarkable ability to learn new skills. Hardware processors, in contrast, never learn. A good manager improves his team by assigning team members tasks that balance difficulty with their abilities. If enough time is available, the manager should schedule activities giving each team member a chance to learn new skills, thus improving the abilities of the team. The scheduling algorithm for a team-oriented process program should take this into account.

3.1.3. Basic Levels of Operation

Computers and human beings differ considerably in their basic level of operations. A program appropriate for a computer is very different from a program appropriate for a human being. For example, when we instruct a computer to add a list of numbers, we must provide a detailed algorithm. In contrast, for a human we would give just a single instruction: add the following array of numbers. The basic level of instructions for the computer (adding and comparing numbers) is lower than the basic level of instruction for a human being. The basic level of operation will also vary between humans. For example, we might tell a child to add by adding the ones' column and writing the result below the ones' column, and any carry above the tens' column then adding the tens and so on. However, anyone past elementary school age would be insulted by such detailed instructions. Similar differences exist in software engineering. A team of experienced professionals needs less instructions than a team of college sophomores. Differences will also exist within a specific software engineering team.

This leads to several important conclusions. The nature of an appropriate processing language for human agents is considerably different than that needed for hardware. Moreover a process programming language must provide a great deal of flexibility in defining operations because basic operations will vary widely from project to project and team to team.

3.1.4. Pro-Active Processors

Human beings are active rather than passive processors. When a hardware processor is asked to perform a particular job, it will perform that job, and no more. A hardware processor asked to compare two module interfaces for compatibility will do exactly that. A human team member asked to compare two module interfaces for compatibility may determine that the modules are indeed compatible, but may make additional observations. He may notice that some of the data structures are poorly handled, or may notice that the module is very similar to one she used in a previous project. Additional information can arrive unexpectedly from any activity. Team members may also go further than instructed. A team member told to fix the bug mentioned in a particular bug report may notice a second bug and fix it. These activities may be desirable, or may be undesirable, but in either case they will occur, and a team-oriented process programming system must handle them.

Because results of an operation are never completely predictable (since humans are unpredictable), the programming language must have a high degree of dynamism (i.e. the language and system must be capable of changing and responding dynamically during run-time). In particular, the language and system must allow the program to be modified at run-time, as actions may occur which require changing the program during mid-execution.

3.1.5. False Failures

When a hardware processor fails in a task, there is either something wrong with the hardware, the program, or the data. In contrast, humans can fail unexpectedly without "real" reasons. If the same instructions are given to another team member, they may perform the task correctly (or at least differently). In fact giving the same task twice to the same team member may provide different results. Problems may occur in which the specified activities are correct yet fail because a team member makes a mistake.

This has serious implications for the meaning of program correctness. We may be able to prove that a particular process program is "correct" meaning that in the ideal case, it will run correctly, but we can never assume that a "correct" program will always run correctly. False failures also mean that the system must be able to change dynamically, responding to the incorrect execution of a correct program.

3.1.6. Creativity

One special characteristic of human beings is their creativity. This presents special problems for process programming because creative processes are beyond our ability to specify explicitly. For the foreseeable future, computers will be capable of only very limited modeling of creativity.

This has important implications for our process programming language. We can either choose to restrict the creativity of the project personnel, allowing them to only perform actions fully specified within the language, or we can assume that the language will not fully cover all the actions of the processors. This second approach requires a fairly large-grain approach to process programming. We can program to the point at which team members use creative processes and can specify the results that we expect from those creative processes, but we cannot specify beyond that point; we cannot specify how the creative actions themselves are to be carried out.

3.1.7. Morale

Hardware machines do not quit because they do not like their work. Hardware machines also do not become less productive when they feel neglected or unappreciated. In contrast to machines, humans have thoughts, feelings, and egos. These all need to be taken into account by project management, and can have a variety of effects on process programming language design.

Ego, for example, can affect assignments and scheduling. A hardware processor will not complain if another processor receives a more important or higher visibility assignment. Human workers may complain or may have their feelings hurt.

Many human workers like assignments with creativity and challenge involved. We may be able to program out much of the creativity of assignments, but doing so may cost us our best programmers as they move to companies offering them more satisfying jobs.

We can treat human workers as machines, giving them step-by-step instructions, however, most humans like at least some control over their work environment. We need to give them flexibility in the execution of tasks, within the limits necessary for task completion. We cannot treat highly educated programmers as auto assembly line workers[2]. Process programming must find a good balance between the control necessary for good project management, and the flexibility which keeps morale high.

3.1.8. Communications

Communication abilities differ considerably from hardware processors to human processors. In some respects, a

2. In fact, more and more, American managers are learning that auto assembly line workers should not be treated as just "assembly line workers". The current trend is toward giving more responsibility, allowing workers to have more control and thus take more pride in their work.

hardware processor is processing intensive (that is, it works well with tasks which are both well specified and require intensive amounts of processing). In contrast human processors are communication intensive (they may not perform as well with processing intensive tasks, but they perform relatively well with communication and in fuzzy, ill-specified tasks). Consider for example the amount of specification which must occur for two hardware-based processes to communicate. The exact information, and the exact format of the information must be specified. Timing between the processes must be specified before the processes begin execution. Handshake protocols must be determined for coordination. Finally, communication breakdowns must be considered, and alternatives planned. In contrast, for communications between two human-based processes, a manager can simply specify, "I expect that you two will keep in touch." This implies that although the exact nature of the information exchange needed between participants is not known before hand, the humans can determine what is needed at runtime. It implies that human beings are naturally adept at communication between themselves, and that they can handle handshake and communication protocols without requiring the manager's help. The humans can also determine when a breakdown has occurred (for example if one process bogs down) and can either correct the situation or cause an interrupt (i.e. notify the manager or system) asking for further guidance.

Of course, sometimes more information is helpful for human communications. In many cases, the manager should plan out in advance which information should be passed between two different processes. Similarly between different groups within an organization, some form of communication protocols should be specified.

A team-oriented process programming language should provide a method for the manager to determine the amount of specification necessary. It should also allow the option of merely specifying that information should be communicated with the understanding that the humans executing the process will determine the actual information and method of communication on their own.

3.2. Team-Oriented Process Characteristics

In addition to the underlying execution agents, team-oriented process programs also differ from application programs in many other important respects.

3.2.1. Macro vs. Micro Time Frame

The typical application program runs in a very short time frame. Even for large database programs which may run for months or years, the individual time to handle a given transaction is relatively short. In contrast, team-oriented process programs can take days or months just to get through a simple program loop (consider for example a define requirements, review, and redefine loop). This macro time frame forces us to look at the program and timing considerations differently. While micro time frame problems are concerned with the O-notation of a program, and whether a particular loop is executed in $O(n)$ or $O(n^2)$, macro time frame programs must be concerned with more specific estimates. For a team-oriented process, whether a particular loop in the program is executed just once, or more than once is a vital concern (because a second iteration through the loop may represent weeks, or perhaps months of additional work).

3.2.2. Operation Timing Considerations

A similar problem occurs in determining the amount of time a particular activity will take. In most application programs the execution time of individual operations (as opposed to algorithms) is rarely considered, although some thought may be given to the use of processor-intensive operations vs. more moderate-cost processor operations (e.g. integer operations over real operations). However in team-oriented process programming, because operations can take days to execute, the timing information is much more important. The fact that a particular procedure or operation will take 2 weeks instead of 3 days is critical. A related problem occurs because the amount of time to perform an operation is rarely known with certainty. With typical application programs, we may be able to determine that an integer operation will take 8 msec and a floating point operation will take 25 msec. The time of team-oriented operations is much less precise, and may prove to be highly inaccurate (and processor dependent).

3.2.3. Unpredictable Behavior

Team-oriented process programs are not fully predictable. This is partly due to the human execution agents and is partly due to the off-line elements of the process. The process is neither completely logical nor completely specifiable. Therefore, we can never fully depend upon our expectations. Any part of the process program can derive unexpected results or may cause unwanted side effects. Any given activity can cause events which require a complete rewrite of the program. In this kind of an environment, dynamism is essential.

3.2.4. Running without Testing

One major difference between the development of application programs and software development process programs is in the ability to test. This problem is one effect of the macro vs. micro time frame difference. A typical application program is validated by both static analysis and dynamic analysis, including running the program on actual data. However, because of the amount of time and resources taken to execute software development processes, actual execution of a team-oriented process for testing purposes is impossible. Testing the process program would basically require developing the software twice, once for testing, once for the actual run. In fact (excluding reuse) a software development process will be run only once. Instead we must depend heavily upon static analysis, simulation, and the reuse of previously used processes. Even under the best assumptions, errors will slip by. Thus during the execution of the process program, we need to expect that errors will occur, and we need to provide a method for dynamically correcting the program and continuing execution.

3.2.5. On and Off-Line Components

A software development process contains many elements very different from traditional application programs. The objects used in the software development process, for example, may exist either on the computer or off-line. In contrast to the typical application program in which all components are on the computer and are fully specifiable, some elements of the software development process are not quantifiable (e.g. how much creativity an activity will require), or only quantify poorly (e.g. social relationships on a software engineering team). The process programming system needs a method for specifying both on and off-line objects.

3.2.6. Unplanned and Underplanned Activities

Software development processes are rarely fully planned out before execution. Sometimes, for example, not enough information is available at the beginning of a project to fully plan all the later stages. Some development models, like the spiral model [Boehm 1990], use early activities to generate information used to plan later activities. During the execution of the process, parts of the process may be completely unplanned, or may contain an underplanned skeleton of the actual process. A team-oriented process programming system should be able to handle both unplanned and underplanned parts of programs. The system should be able to both begin execution of an incomplete program and provide some analysis of an underplanned program.

3.2.7. Management Issues

A software development process is managed in a different manner than an application process. Management from both within and outside of the process may affect the process program. One process inside the process program may be set up specifically to manage a second process or a group of processes. In hardware-based processes this management is rudimentary at best (for example, an operating system process manages the processes under it). In contrast, management by human executed processes can be quite complex. A human manager is capable of stopping and modifying any processes under his jurisdiction at any time. Management outside of the scope of the process program may also step in, adding or removing resources, modifying project goals, requiring changes to the process program, or possibly terminating the process. A team-oriented process programming language must provide a richer method than those available in application languages for the specification and handling of the management of processes. Process management also requires dynamism in the process programming system to allow modification of running processes.

4. Critical Issues in Team-Oriented Process Programming

We have identified three key issues which we believe must be addressed in order to make team-oriented process programming a reality. They are dynamism, specification of resources, and customization and flexibility. We also believe that user interface issues must be addressed in order to build a usable system.

4.1. Dynamism

The system and language must allow for a high degree of dynamic change. Because new program code will be created within the program, the language must treat programs as objects. For example, the team process program might specify that the manager should develop a plan for determining requirements. In this case, the requirements part of the process program will be the output of the planning part of the same process program. The system and language should also allow creation of types dynamically. During the execution of one of the activities, a processing agent may decide that a new type needs to be created. Using the previous example, the manager might decide to create a new object type specifying the form that a requirement should take. This decision would be made, and the new type created, as the process program is being executed.

The system must also provide a means for changing running process code. As we have argued in the previous section, change may be necessary for a variety of reasons: the team process program is untested and may contain bugs, outside changes from management may require changes in the program, the team members may not act as expected, requiring changes to the program. These changes occur in a spectrum ranging from changes in the resources available to a running program to outright cancellation of part of the process. The system must allow these changes to occur without stopping the process, recompiling, and restarting execution from the beginning the program.

4.2. Specification of Resources

A team-oriented process programming system must provide a method for specifying resources. This is particularly evident when considering the processors needed for a given activity. Traditional programming languages (e.g. Ada, FORTRAN) do not allow the programmer to specify the characteristics needed by the processor assigned to a given procedure or operation. However, because the abilities and experience of team members vary so widely, operations in a team-oriented process program should provide processor specifications. For example, an activity to code a printer driver in Ada might require that the team member assigned have a high degree of expertise with Ada and past experience writing i/o drivers.

This concept can be more fully expanded to encompass all resources used in team-oriented processes. A wide variety of resources can be specified including computers, budgets, access to specific programs, availability of meeting rooms, even availability of overhead projectors. For example, in a project in which only a limited number of computers were capable of running a particular compiler the team process programmer might specify which activities required the compiler.

Two forms of resource specifications will be needed. For a given activity the language must allow the team process programmer to specify the characteristics of the resources needed. At the same time, the system should allow the user to specify the attributes of the resources available.

4.3. Customization and Flexibility

As we have previously argued, the needs of teams will vary widely. Different managers will want to use the team process program to specify different levels of detail. The level of detail needed to direct team members will differ depending upon the experience of the team. A team-oriented process programming system should allow the team manager to use the primitive operations which match the needs of his team. For example, the manager of an inexperienced team might want to provide a step-by-step specification of the development of a module specification.

Primitive operations in this case might include: "determine types externally visible" and "determine procedures externally visible." The manager would explicitly specify everything that needed to be done. In contrast, the manager of an experienced team might want a single primitive operation: "develop module specification." In addition to experience, the needs of the project may vary depending on the software development methods chosen. For example, a team using clean room testing techniques might have very different needs than a team which handled its own testing. Managers also might specify operations to match organization specific procedures. In a company using management-by-objectives (MBO) [Anderson 1984], for example, the manager might want to set create data types to represent the objectives of project personnel and specialized activities to represent MBO objective review meetings.

4.4. Human Interface Issues

If a team-oriented process programming system is to be used by both computer experts and novices, a well designed human interface will be essential. Novice users may be aided by the use of graphics and an emphasis on familiar metaphors.

Training time and comfort level for novice users can be improved by the use of metaphors. For example, the Xerox Star and later the Apple Macintosh used the desktop metaphor, allowing users to see the electronic equivalent of their already familiar desk and folders. We want to provide similar types of metaphors for interfacing with the team-oriented process programming system.

5. Other Issues in Team-Oriented Process Programming

In addition to the critical issues which must be addressed in order to make team-oriented process programming a reality, we have identified a number of other issues which, if addressed, can provide additional benefits.

5.1. Planning and Projecting Information

Team-oriented process programs will be used both to plan and execute the software development process. In order to provide maximum utility, a team-oriented process programming system should supply information for planning and projecting. This kind of information is currently provided by management tools like PERT/CPM and Gantt charts. These tools aid a manager by determining how long a project will take, which activities are most important, and which can be delayed without affecting the project completion date. Unfortunately, PERT/CPMs methods do not transfer to team-oriented process programming, because these methods do not handle the conditional and looping programming constructs. Moreover, because the path through a team-oriented process program's loops and conditionals can not be determined prior to execution, we can never provide the precision of the simpler, deterministic (and less realistic) PERT/CPM model. Team-oriented process programs may also include sections which have not been planned, or which have only been sketched out. We must develop new methods for providing timing estimates for team-oriented process programs.

5.2. Communications Protocols

As we have discussed, humans have different communication abilities from hardware processors. In fact, humans can communicate using a wide variety of methods. As discussed in section 3.1.8, a team-oriented process programming language should provide a method for specifying communication between activities. A set of protocols should be developed which allows a team-oriented process programmer to specify both the method of communication and the information to be communicated. For example, with two team members working closely on a design, the team-oriented process programmer might specify that the two team members should communicate informally. No further specification would be given. In contrast, a customer liaison communicating with the programmer responsible for code maintenance might be required to communicate formally. A form specifying the exact information to be exchanged might be provided.

5.3. Management Protocols

Management and organization of teams are key issues affecting the execution of team processes. A team-oriented process programming system should allow specification of the management of the process. Management protocols should be provided which show which management processes are responsible for each activity. Reporting relationships between team process activities should be shown. Information on management control of resources should also be provided.

5.4. Scheduling Algorithms

As we have seen, a wide-variety of factors goes into scheduling of human processing agents. A team-oriented process programming system can provide added benefit for managers if it includes scheduling capabilities. Scheduling assistance might range from completely automated schedulers [Watanabe & Osterweil 1990] to completely manual methods. Schedulers should take into account the different nature of the various resources involved in a team-oriented process program. Humans, for example, will perform inefficiently if switched rapidly from one task to another. In contrast, switching a computer assignment or room assignment from one task to another will probably not cause problems.

5.5. Off-Line Artifacts and Resources

Because a great number of team-oriented process programming deals with artifacts which may be off-line (e.g. manuals or books), a team-oriented process programming system can provide additional benefits if it can track off-line artifacts. These artifacts might be tracked using some kind of library system. Using this system in conjunction with an on-line object management system would allow team members to find any object related to the project.

In addition off-line resources (e.g. meeting rooms, budget) should also be tracked and, if possible, managed by the system.

6. Mechanisms for Team-Oriented Process Programming

In this section we discuss mechanisms to support a variety of the aspects of team-oriented process programming. We describe how an object-oriented system might be used to meet some of the critical issues discussed in the previous section. We discuss how management tools and time management techniques can used as metaphors for the user-interface. We conclude the section by briefly describing how a team process program might interface with another type of domain-specific process program.

6.1. Object-Oriented Approach

As we have seen, the language and system should be customizable. Because object-oriented systems are typically easy to customize, they provide one potential starting point in our search for solution mechanisms. In this section we show that if the idea of objects is extended to cover activities, artifacts, and resources, an object-oriented system can help in all four of the critical issues we have identified.

If both activities and artifacts are based on the same object system, we can treat activities as artifacts. This permits us to treat programs as data, allowing us to handle cases in which part of a team process program creates or modifies another part of the process program. Object-oriented systems also typically provide for run-time creation of new types.

An object-oriented system also provides a simple uniform method for allowing users to customize the system. Projects could easily create new activity description types, artifact types, or resource types which better represent the needs of their project. Specializing types by adding new fields and by combining existing types is a relatively simple operation and should be performable by both computer experts and novices. For an example of a customizable typing and object system similar to the one we describe here see the Object Lens system [Lai et.al. 1988].

Resources can be specified as one of the fields of an activity description object. Specialized activity descriptions types can be setup for specific resources. For example, the project may have a limited amount of space, and may treat meeting rooms as a resource. In general, the location of an activity is not stored. However, the team-oriented process programmer could create a new meeting activity description type which included a meeting room field. During planning or execution, the system would allocate a meeting room for each meeting activity.

Finally basing activity descriptions, artifacts, and resources on the same typing model helps provide uniformity in the human-interface. The interface will appear similar whether working with procedural information, data, or resources.

6.2. Interface Metaphors

Using interface metaphors will help make the team-oriented process programming system accessible and friendly to novice computer users.

The PERT chart [Anderson 1984] is used by managers for planning and controlling projects. It can be used as a powerful metaphor for the manager's interface with the team-oriented process programming system. Managers use PERT charts to show a project's activities and the dependencies between the activities. This is very similar to the information we expect in a team process program, although the team-oriented process programming language provides both automation and more descriptive power. In order to use the PERT metaphor, we need to enhance the PERT diagrams to allow the manager to specify loops and conditionals and to handle the additional information associated with team process program activities (e.g. objects input, objects output, resources). The team-oriented process programming system also tracks activity progress and displays changes automatically. This information is done manually with PERT charts. The Gantt chart, used by managers to provides a more time-oriented view of a project, can also be used as a metaphor for team-oriented process programming.

Because the team process program directs the activities of team members, time and activity management methods are a natural place to look for a metaphor for team members. An interface which has the look and feel of the time management/datebooks widely in use by managers and workers would probably be well accepted. Information from the team process program could be passed to workers through calendars and to-do lists. Activities assigned to a team member could be displayed in that team member's to-do list. Deadlines and milestones could be marked in the team member's calendar. Team members would be notified as activities become enabled. Activities which were nearing their deadline, or activities which had become late might be highlighted on the list and calendar. Hopefully these interfaces will reduce the alienation of users. We want the user to view team-oriented process programming as a new, more efficient, automated way of helping them do what they are already doing, not some nefarious scheme to have a computer control their life.

6.3. Interface with Domain-Specific Process Programs

As we have previously mentioned, we believe that a team-oriented process programming system should be interfaced with other domain-specific process programs and languages. The team-oriented system could provide support for the needs of teams and specify interaction between human processing agents, while other domain-specific process programs would provide support in other areas.

For example, the REBUS program [Song et.al. 1990] written in the APPL/A process programming language [Sutton et.al. 1990] is a process program written for developing functional requirements. REBUS describes the form of the requirements and the relationships between the requirements in detail and provides methods for creating and modifying requirements. REBUS does not supply such information as the team-members to develop the requirements and the expected interaction between team members. This information could be supplied by the team process program. Additional information in the team process program, not specified in REBUS, would include the relationship of the REBUS process to the larger software development process and the information needed to create the requirements (especially off-line artifacts like user interview results).

7. Uses of Team-Oriented Process Programming

In this section we discuss how team-oriented process programming might be used by both management and team members.

7.1. Management

Team-oriented process programming has the ability to aid management in each of the five traditional management functions: planning, organizing, staffing, directing, and controlling. Team-oriented systems are especially well suited to support the planning, directing, and controlling functions.

7.1.1. Planning / Process Programming

Planning is arguably the major management function. Planning corresponds directly with process programming. During the initial planning stages we would expect considerable process programming.

Programming the team-oriented process is roughly equivalent to developing the operational plan for the project. During planning the manager determines what activities need to be completed in order to meet project goals. Resources needed for each activity are specified. High-level activities are elaborated into sets of more specific activities. Dependencies between activities are identified and the activities are ordered. These steps are the same steps that the team-oriented process programmer takes. Process programming, however, provides significant advantages. The specification is formal and includes the full power of a programming language. More importantly, as we shall see in the subsection on "Directing" the process program is executable.

The high-level abstractions of team-oriented process programming make it easy for managers to be intimately involved in process programming and planning at this stage. In addition to working with the team-oriented process, the manager should also determine which domain-specific process programs will be used. A manager might, for example, choose the REBUS process program [Song et.al. 1990] to handle requirements, and an object-oriented design version of the DEBUS [Song & Osterweil 1989] design process program to handle requirements. These domain-specific process programs might be chosen from a library or might be custom built (or at least modified) for the project. These programs provide detailed descriptions of how specific activities are to be carried out. They enforce policy at levels below the resolution of the team-oriented process programming language. At the same time, they require considerably more resources to develop than the team-oriented program. They also require much greater programming expertise. In contrast to the team-oriented process programs, these domain-specific processes cannot be written by management. They will probably also be less project specific. Because of this, we envision a pre-written library of domain-specific process programs. A company could, for example, embody their process for developing modules in a special process program. This program would be used by many projects. These domain-specific process programs should be designed to interact with the team-oriented process program.

Using the team-oriented process programming system, the manager could develop a number of competing plans for the project. A wide variety of analytical tools could be used to compare these plans. The formality of the team-oriented process program eases qualitative comparison between plans. The computer can, for example, easily determine the resource requirements of each plan. Budget and risk analysis can be conducted. The tools available should include traditional management techniques (e.g. critical path calculations) and tools developed for computer programs (e.g. concurrency analysis).

7.1.2. Organizing

While organizing the project is not directly tied to process programming, the process program will affect the organization of the team and the organization will affect the process. Ideally a process programming system should combine organization capabilities with the process program.

Tools can be set up to describe the team structure. The hierarchy and grouping can be shown graphically. Lines of

reporting and responsibility should be shown. These capabilities should match directly with the management protocols mentioned in in section 3.2.7. Organizational positions should be linked with the personnel resources listed in the activity descriptions of the team-oriented process program. Once these interconnections have been determined the system can aid in the development of position descriptions. The process program's activity descriptions can then be used to provide estimates on the abilities and experiences needed for each organizational position.

7.1.3. Staffing

Much of the management staffing function is peripheral to the process program. However, staffing is directly involved in the allocation of resources (especially human resources) to activity descriptions. A process programming system should be capable of helping out with both the peripheral and more direct aspects.

The team-oriented process helps directly with the filling of organization positions within the process program. A database of company personnel available can be matched to the positions. For a given position the system summarizes the skills and experience needed by checking on each of the related activity descriptions. This summary is compared to the available personnel and ratings are developed showing how personnel and positions match. Timing and scenario information can be combined to determine percentages of work time each person will be needed. For positions which can not be filled, the system can summarize requirements and pass the information on to the personnel department.

Staffing also includes recruiting, training, and evaluating personnel. The system's skills and experiences summary for each position can be used as an aid in recruiting for each position. As we shall see in the section on "Uses of Team-Oriented Process Programming: Team Members" (Section 7.2) the team-oriented process can also act as a tool for helping new project members understand their position within the process. Finally team-oriented process programs may be annotated to aid in the evaluation of personnel.

7.1.4. Directing / Process Execution

After the project has been planned, resources organized, and the project staffed, execution of the process begins. During process execution, the team-oriented process program helps the manager in directing the project. The system delegates tasks to team members as determined by the team process program. This has significant advantages over current methods. The use of team-oriented process programming provides a uniform method for communicating tasks. Moreover, as we shall see in the section on "Team Members" each of the team members can study the context in which the task has been given. Delegation of activities can be fully automated, reducing the burden on management. If desired, the system can notify the manager of activities which may be enabled, and wait for the manager's approval before activation.

The team-oriented process program also aids in coordination of tasks. Using the communications protocols of the language team members are informed of team members with whom they should be coordinating. As we will discuss in the "Team Member" section, the system also provides easy access to information on related activities, objects, and the personnel assigned to them. This should help further coordinate the team. Communication is facilitated by knowing who to contact and how to contact them.

7.1.5. Controlling / Process Monitoring and Modification

Controlling the software development process includes the ability to monitor and modify the process program. Because the team-oriented process program is actually executing, the information available is always current. This is in sharp contrast to traditional methods in which the status indicated on the project plan is always behind the actual process. Tools available for process monitoring should include: direct graphic representation of the team-oriented process program indicating which activity descriptions have been completed and which are active (this is essentially a graphical representation of the line-by-line execution of a program), status indicators for management connected to instrumented team-oriented process programs and domain-specific process programs which indicate

completion of milestones and other data predetermined as important, indication of how the process is meeting planned deadlines and automated notification when particular parts of the process fall behind schedule. Other process programming tools should be able to access information to keep management assessed of risk levels, budget levels, and resource needs.

Equally important is the capability of modifying the running process. If the program is not executing as planned, the system must also provide capabilities for modification of the running program. The manager should be able to derive new versions of the executing process program. The system should support planning and comparison of different modifications. The system should aid the team in switching to modified versions of the process program.

7.2. Team Members

The team members should also benefit from the use of team-oriented process programming. Special care must be taken to insure that process programming acts to enhance job satisfaction and performance, and to make sure that process programming provides workers with significant benefits at little cost.

If properly designed, a team-oriented process programming system should allow the team members to spend less time worrying about details and more time working on creative activity. The system should act as a well designed time-management system, providing details of the team member's current activities. Because the software development process is running on the computer, the process programming system can go much further than a time-management system however. The system should provide easy access to the tools and objects which the team member needs to access. The system should also aid the team member by providing easy context switching between the many activities which he is currently working on.

The process program can also act as an extensive database organized both by the activities and the objects that he is currently working with. The worker should be able to use the system to study how his activities fit into the activities of fellow team members. Similarly the worker should be able to see which activities and which co-workers have affected a particular object. He should also be able to study how that object is related to other objects in the system. This should facilitate communication in the group. A team member can easily determine who needs to talk to whom about a specific problem. Similarly, a team member can determine who he should notify if he needs to change a particular object.

The database is also a valuable tool for helping new employees understand the project. A new worker should be able to see the activities he needs to work on. The system would also allow the worker to explore the larger project vision. This should reduce the time needed to bring new employees up to speed in a particular project. It should also improve morale by helping new employees see how their work fits in to the project.

The system should also provide the same functionality for workers that it does for managers. A worker should be able to develop subprocedures and use the system to track his activities. If warranted, the worker might act in the management role, providing support and direction for a set of workers.

The process programming language also provides a mutual language for communication of process ideas between management and workers. If a worker sees a problem with the current software development process he should be able to create a modified version and send that version to management. As with manager's initial planning phase, tools would be provided to aid in comparison, and to highlight differences (both good and bad) between the current program and the new program.

7.3. Subteam Coordination

The system should aid in coordination between wide-ranging subteams and team members. As previously mentioned, the system can act as a database for activities and objects. The system should also provide appropriate versioning and configuration control for subteams. If written correctly, the process program should provide information allowing the system to determine which versions should be viewed by particular subteams or team

members. A quality assurance subteam, for example, probably needs to work with the final code version, and does not need to see various test versions and prototypes. In contrast a team member working on a specific module would want to have access to a wide number of versions including versions with test stubs and test drivers. The system should also use the process program to determine which versions should be used for particular activities, and should provide tools to help team members understand object versions and their relationships.

The system should also help manage the interface between various groups. Activity information should be hidden and managed in a manner similar to objects. In all cases the objective is to use the process program to manage the amount of information which must be studied.

8. A Vision of Team-Oriented Process Programming

As part of the design of a team-oriented process program, we have developed a number of sketches showing how a team-oriented process program might look. Figure 1 (page 15) is one of these sketches. It shows the windows of Mike Brookings as he interacts with a team-oriented process programming system. Mike is managing a team of seniors in our upper-division software engineering project class; the process shown is not meant to represent an industrial process.

Windows 1 and 2 (upper and central right) show how the system helps Mike to manage his work. Window 1 "Brooking's To-Do List" presents a list of activities and their due dates. Activities completed are marked by checkmarks while those which are especially urgent are highlighted. The arrow icons in the right column of the window and the icons showing a group of windows in the far right column are used for navigation and context switching and are described below, at the end of this section. Window 2 is a calendar showing due dates and completed activities. Meeting activities will also be marked here.

Window 3 (upper left) shows part of a currently executing team process program. This window can be used to monitor the progress of team process program. Nodes which have a cross hatch pattern (for example "Plan" and "Initial Requirements" in the left end of the window) have been completed. Activities that are outlined ("DD Draft M1", "DD Draft M2", and "DD Draft M3"[3]) are currently executing. Window 4 shows an alternative team process program. This program is currently inactive, but Mike could use the system to activate it and notify team members of changes to the process.

Window 5 is an information window associated with the "Develop Detailed Draft Model 2 Activity" (abbreviated "DD Draft M2" in window 3). It shows timing information, the expected inputs and outputs of the activity, and project personnel associated with the activity.

Windows 5 and 6 are personnel description windows showing contact information, assignments, and abilities of two of the team personnel.

A team-oriented process programming system could treat the components of the team process program as a hyperweb of objects. Team members could navigate between them using a hypertext-like system [Conklin 1987]. For example, by clicking on the various arrow icons in the right column of his "To-Do List" (window 1), Mike can bring up information windows on each of the activities. Clicking on the arrow icons in the "Objects Required" list of window 5 would bring up those objects. Clicking on the activity nodes in the team process program windows (3 and 4) would bring up information on those activities and subdefinitions for those which are procedure calls[4].

---

3. "DD Draft M1", " DD Draft M2", and "DD Draft M3" represent three competing detailed designs, each based on a different premise. The team will meet in the "Integrate/Select" meeting activity and select and/or integrate the three designs. The activities following this meeting develop a final version of the detailed design.
4. Activities nodes which have further elaboration in the team-oriented process programming language have *'s in the lower right-hand corner. For example the "DD Draft M1," "DD Draft M2," and "DD Draft M3" activities are all defined by additional team-oriented process program code not shown here.

A team-oriented process programming system could also provide context switching (see the XEROX Rooms system [Henderson & Card 1986]). The icons representing a group of windows in the far right column of window 1 allow Mike to store and restore context information for each of his activities. The current set of windows and tools are used by Mike to "Develop Revised Plans" and "Monitor Progress." Clicking on the context icon to the right of the "Develop Travel Functions Module" would hide the current windows and bring up the set of windows and tools Mike was using to develop the design of the Travel Functions module. Thus we can use a team-oriented process programming system to tie context work in with process management.

9. Current Work

We are building a team-oriented process programming language and system based on the ideas in this paper. The Teamware language is designed for writing programs for human processing agents, and the Project Assistant system is designed to aid in the execution of these programs. The system is designed for widespread use by both those with programming backgrounds and those with modest programming skills. We hope that team-oriented process programming will make many of the advantages of process programming available to a much wider audience.

10. Summary

In this paper we have developed the idea of team-oriented process programs. We have explored key characteristics of team-oriented process programming. We have identified critical issues which must be solved and have briefly explored potential solutions. We concluded the paper by describing some of the advantages of team-oriented process programming and by providing a vision of what a team-oriented process program might look like.

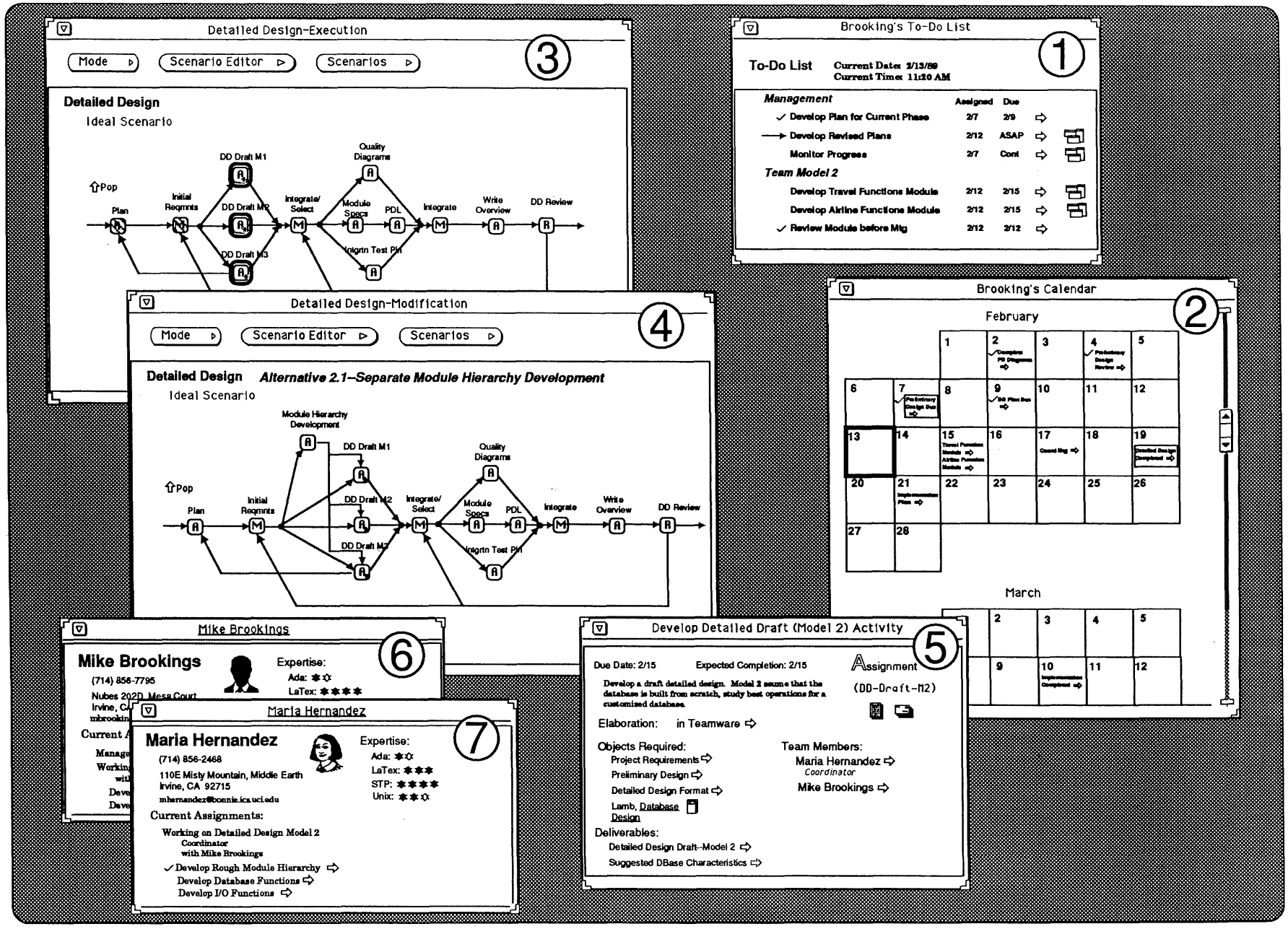

Figure 1: Sketch of a Potential Team-Oriented Process Programming System

Anderson 1984 Anderson, Carl R., *Management: Skills, Functions, and Organization Performance*, Wm. C. Brown Publishers, Dubuque, Iowa, 1984.

Boehm 1990 Boehm, B.W., "A Spiral Model of Software Development and Enhancement." In Thayer, R.H. *Tutorial: Software Engineering Project Management*, IEEE Computer Society Press, 1990.

Conklin 1987 Conklin, Jeff, "Hypertext: An Introduction and Survey," *IEEE Computer*, September 1987.

Curtis et.al. 1988 Curtis, Bill, Krasner, Herb, and Iscoe, Neil, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, Volume 31, Number 11, November 1988, pp. 1268-1287.

Grudin 1988 Grudin, Jonathan, "Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces," *CSCW 88: Proceedings of the Conference on Computer-Supported Cooperative Work*, Portland, Oregon, September 26-28, 1988.

Henderson & Card 1986 Henderson, D. Austin, Jr., and Card, Stuart K. "ROOMS: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface," *ACM Transactions on Graphics*, Volume 5, Number 3, July 1986, pp. 211-243.

Lai et.al. 1988 Lai, Kum-Yew, Malone, Thomas W., and Yu, Keh-Chiang, "Object Lens: A Spreadsheet for Cooperative Work", *ACM Transactions on Office Information Systems*, Volume 6, Number 4, October 1988, pp. 332-353.

Olson 1989 Olson, M.H. (Ed.), *Technological Support for Work Group Collaboration*, Lawrence Erlbaum Associates, Inc., 1989.

Song & Osterweil 1989 Song, Xiping, and Osterweil, Leon J., *Debus: a Software Design Process Program*, Arcadia-document, UCI-89-02, April 1989.

Song et.al. 1990 Song, X., Maybee, M., Osterweil, L.J., and Heimbigner, D., *REBUS: A Requirements Specification Process Program*, Technical Report UCI-90-17, Department of Information and Computer Science, University of California, Irvine, 1990.

SPW4 1988 Tully, Colin (Ed.), *Representing and Enacting the Software Process: Proceedings of the 4th International Software Process Workshop*, ACM Press, 1989.

SPW5 1989 Perry, Dewayne E. (Ed.), *Experience with Software Process Models: Proceedings of the 5th International Software Process Workshop*, IEEE Computer Society Press, 1990.

Sutton et.al. 1990 Sutton, S.M., Heimbigner, D., and Osterweil, L.J., "Language Constructs for Managing Change in Process-Centered Environments," in *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, Irvine, California, December 1990.

Watanabe & Osterweil 1990 Watanabe, Gen and Osterweil, Leon J., *Software Process Scheduling Based on Process Programming*, Arcadia Technical Report CU-90-02, August 14, 1990.